Derek Gerstmann (#9722369 - dgerstma@u.washington.edu)

Derek Gerstmann - 9722369

Users Manual

Introduction: Welcome to *Box Collector*, a simple 2D game somewhat similar to a popular circle-munching ghost-eating arcade classic.

Goal: To become a master champion of *Box Collector,* you must make it through each and every level. Each level tests your ability to locate and pickup randomly placed boxes located throughout each level, before running out of time. All boxes must be picked up before time runs out, or you will lose a life and be forced to replay the level until either you succeed, or run out of lives.

Display: The game provides you with two views of each level, the global view and the main view (see ***Interface***).

The *main view* showcases your vehicle and the are closely surrounding it (including all boxes and obstacles). You can change the amount of area visible in the *main view* by issuing a '***zoom in***' or '***zoom out***' command (see ***Commands***).

The *global view* provides you with an all-encompassing vantage point that allows you to identify your current position with respect to the entire level. As you move your vehicle around, a white square will be updated to represent the viewing extents of the *main view*. Note that although the *global view* shows the entire level, it does not show any of the boxes in the level. That would be cheating.

Game Play: After the brief logo introduction is complete, the game will immediately start on the first level and game play will begin.

As soon as you can see your vehicle, the timer has already started, so get going!

Start searching for boxes. Increase your velocity and turn towards the first visible box. If no boxes are visible, navigate your vehicle towards the edge of the main view and continue your hunting.

If your vehicle runs into an obstacle, or wall, your vehicle will be forced to stop, and you'll have to maneuver around the object.

If you find a box, run your vehicle over it to pick it up. Continue picking up and searching for boxes until you've cleared the level.

Keep track of your time! If the timer counts down to zero, you will lose a life and will have to replay the current level.

If you run out of lives, the game will end, and you'll have to start over from the first level.

Derek Gerstmann - 9722369

Interface:

In addition to the main and the global views, the game also has several other useful areas located on the graphical user interface (GUI)

The first area, the game status area, continuously reflects the current state of the game. This includes such items as the amount of time remaining, the speed, and direction of the your vehicle, and the number of lives that you have left.

The second area allows you to issue movement commands as an alternative to the keyboard shortcuts. Use the arrow keys in the direction pad to control your vehicle just like you would with the keyboard.

The third area provides you with general game information and has buttons allowing you to pause and exit the game. General messages will appear in the middle box at the top.

The general layout of the GUI, including the control panel, status area, and global areas, is illustrated below. You may wish to familiarize yourself with the GUI before playing the game.

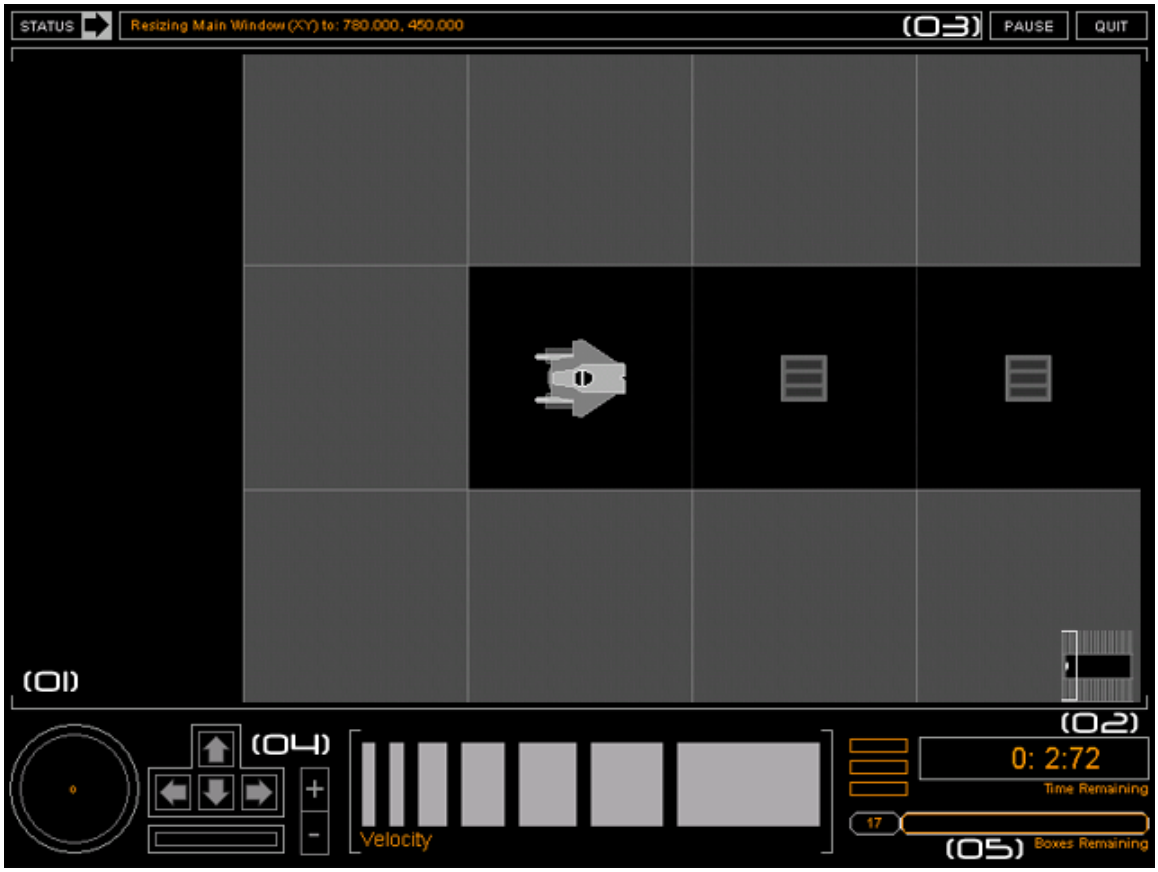

Image 1.0 –Layout of the Graphical User Interface

	Title	Description
1	Main View	Main view of the game. Your vehicle and any close objects are shown in this area.
2	Global View	Global view of the game. Everything except the boxes are shown in this area.. A white box reflects the viewable area of the main view.
3	Application Status and Control Area	Provides information not covered by the game status area. Also contains controls for pausing and exiting the game.
4	Control Panel	Contains controls for issuing movement commands and controlling your vehicle.
5	Game Status Area	Includes the game's timer, the speed, and direction of your vehicle, and the number of lives you have left.

Table 1.0 – GUI Layout

Commands: All commands are issued through keyboard shortcuts, or via the control panel area of the GUI. A list of the available keyboard shortcuts can be found below.

Key	Command
Up Arrow	Increase player's velocity.
Down Arrow	Decrease player's velocity.
Left Arrow	Decrease player's rotation.
Right Arrow	Increase player's rotation.
-	Increase the zoom factor for the main view.
+ / =	Decrease the zoom factor for the main view.

Table 2.0 – Keyboard Shortcuts

An additional set of debug commands has been included in this release. Feel free to experiment with these commands, but do not rely on them to improve your game playing abilities!

Key	Command
p	Toggle the bounds of the player's vehicle.
[	Goto the previous level.
]	Goto the next level.
q	Exit the game.

Table 3.0 – Debug Keyboard Shortcuts

Objects:

There are three types of objects for the game, your vehicle, boxes, and obstacles. A brief summary of each object and its corresponding graphical representation is given below.

Your vehicle is a high-performance grav-tred ship manufactured by a government subsidized, top secret research project. As such, take great care and avoid any obstacles you may come across. (You wouldn't want to damage your ship, now would you?) A picture of your vehicle is given below.

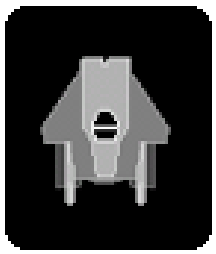

Image 2.0 – Player's Vehicle

Obstacles are composed of primitive shapes. If your vehicle runs into an obstacle, it will come to a complete stop. It's probably best just to avoid obstacles if you can. An image containing several of these annoyances can be seen below.

Image 3.0 –Obstacles

The most important object in the game are the Boxes which you need to collect. Once you find a box located (hidden throughout each of the levels), run your vehicle over it to pick it up. Make sure you collect all of the boxes, before time runs out, or you will lose a life and be forced to replay the current level.

Image 4.0 – Boxes

Known Bugs:

You should be aware that the development of this game was driven by strict guidelines and deadlines. The developers did their best to minimize the number of errors that were included in this release, but several known bugs do exist. These are listed below.

Bug	Description
Circle Pad	The circle pad in the control panel is currently inactive.
Space Button	The space button under the directional controls in the control panel currently does nothing.
Difficulty	There is currently no way to set the timer difficulty setting.
Resizing	Resizing the games display window may or may not crash the application. This is most evident during the intro.

Keystrokes	Not all keystrokes seem to have the same repeat value, and thus some keys may not respond in the same amount of time as others.
Collision Resolution	There are some instances where the player's vehicle can get stuck inside of an obstacle. This is rare, and seems to show up only when the vehicle is continuously being bumped up against and obstacle, and starts to turn.

Table 4.0 – Known Bugs

Derek Gerstmann - 9722369

Project Report

Design:

For the design of my game, I decided to follow the example given in class and separate the core graphics library from the logic of the game engine, and developed each of these as individual projects.

Graphics Library:

For the core graphics library, I built upon the hierarchy presented in class and extended the described functionality to suite my interests and needs.

I also tried to abstract the classes themselves in hopes that they might be used for a more elaborate project in the future.

One particular abstraction was the specification of 2D objects which derived from a generic abstract “Object” class. By specifying the abstract interface for a generic object, I was able to make the derived 2D objects much simpler, and hopefully allow a set of 3D objects to be built (some time in the future) that use the same generic interface.

In order to support collisions and moving objects, I created an abstract “2D Particle” object, that I derived from the generic “2D Object”. This allowed me to avoid multiple inheritance (and subsequent run time type interpretation) and provided all of the derived “2D Drawable”, and “2D Fillable” objects with the mechanisms needed to move and collide. Since not every object would be required to do these tasks, the behavior dictated by the “2D Particle” can be optionally enabled and disabled.

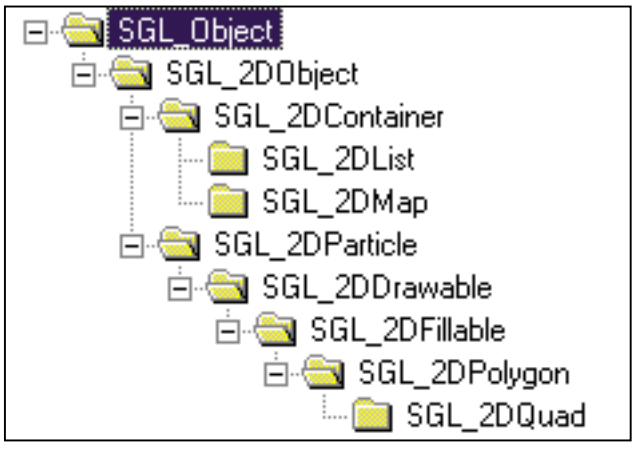

Image 5.0 – Basic Object Heirarchy

I also created an abstract “2D Container” class provided a means of grouping an arbitrary number of “2D Objects”. Derived from this class are a “2D List”, which implements a dynamic list of objects, and a “2D Map” which provides a means of spatially sorting a set

of “2D Objects” based on their world positions with arbitrary precision. Since the “2D Containers” are themselves “2D Objects”, “lists of lists”, “lists of maps” and “maps of maps” can all be created easily.

Another optimization that I made was the use of OpenGL display lists to cache any static geometry, object attributes, or transformation information. Furthermore, by maintaining a list of points (or vertices) for each object, I was able to dynamically create and destroy the display lists in the event that either the geometry, attributes or transformation information needed to be updated.

Another extension that I focused on during the development of the core graphics library was the separation of objects and geometry. In order to efficiently use the available system resources, I created an abstract “2D Geometry” class which can be referenced by any “2D Object”. This allows multiple objects to use the same primitive shape, but only requires a single set of vertices to be stored in memory.

I also created wrapper objects for each type of geometry to allow for easy object construction. For example, a “2D Line” object creates it’s own associated geometry (“2D Line Geometry”) from which other objects may be specified.

Game: For the development of the actual game, I decided to rely on the graphics library to provide most of the functionality, thus requiring only the necessary event handlers, *GUI* operations, and state machines to be implemented.

Event Handlers: I decided to make the *Event Handlers* have very little responsibility and, instead, act as the glue connecting the *GUI* with the actual game engine. As a result, the *Event Handlers* themselves were trivial to implement.

One problem that I did encounter when I was implementing the *Event Handlers* was the detection of multiple key strokes. As it turns out, the GLUT wrapper that was being used did not explicitly state all of the keys that were being pressed at any one time, so I decided to keep track of this information in a keystroke buffer.

Whenever a keystroke event was registered, the value of the key is inserted into the keystroke buffer, to later be processed during the update associated with timer event handler.

GUI: For the *GUI*, I created an initial layout that I modeled after a design I created in a popular paint program. As development continued, many different features were fine tuned, and adapted to support the changing features of the game.

Some items, however, did not get corrected before the time of release. For example, the circular control pad in the lower left corner of the window is not active. Likewise for the bar underneath the direction pad buttons (see ***known bugs***).

Design:

The game engine itself is composed of a finite state machine, a player entity, a level map, some graphical objects and state methods.

The player entity keeps track of the necessary player information (number of lives, current level, preferred difficulty setting, etc.) It also provides methods for altering the velocity and direction of the player's vehicle. The actual representation of the players vehicle, however, is maintained by the game engine, so that only a single instance of geometry is required (thus allowing for the possibility of multiple players).

The level map is a storage class for all of the objects associated with a level. This class is initialized from a data file that is read off of disk and parsed. It also maintains all of the information associated with the level, including the initial player position, and the values for the different timers.

The state methods provide a means of switching the current context or mode that the game engine is currently in. The basic state transitions are discussed below.

States:

When the game starts, it jumps into a ***play intro*** state in which a series of images are shown for an arbitrary amount of time, before the game is switched into a ***new game*** state.

In the ***new game*** state, the game display is initialized, a GLUT window is created, and the current level is read from the data file. The game's timer is initialized to the value associated with the default difficulty setting (currently Normal) and the game moves into a ***playing game*** state.

During the ***playing game*** state, the level and all objects are continuously redrawn, and the user is given direct control over their vehicle (see ***commands***). The *normal view* follows the position of the player's vehicle and a white box in the *world view* is updated to represent the change. The games timer continually decrements.

The user can continue to move throughout the world until any part of their vehicle touches, or collides with, either an obstacle or a box.

If an obstacle is encountered, the vehicle's motion is reduced to a stand still (zero velocity).

If a box is encountered, it is set to invisible, and uncollidable, (essentially removed from the world), and the number of boxes is decremented.

The ***playing game*** state continues until either the timer runs out, or all boxes are collected, in which case an ***end of game*** state is triggered.

In the ***end of game*** state, if all boxes were successfully removed, the player's current level is increased and a ***showing nextlevel banner*** state is initialized. After the "Next Level" banner has finished being displayed, the game shifts into a ***new game*** state.

If the timer equals zero in the ***end of game*** state, the number of lives for the player are decremented by one and a ***showing outoftime banner*** state is initialized. After the "Out of Time" banner state is complete (finished being displayed), the game shifts into a ***new game*** state.

If the number of lives equals zero, a ***showing gameover banner*** state is initialized, before the game resets into a ***showing intro*** state.

Extra Features: In addition to the number of unrequired/advanced features that I was able to include, there is a long list of extra features (in addition to fixing all known bugs) that I would have liked to include, if extra time were available.

Each of these would be relatively easy to implement given the current state of the graphics library. A list of these features has been provided below.

Feature	Description
Sound	Supply a wide variety of sound effects and music custom tailored to each level.
Menus	Provide a method to set the difficulty setting and other options that are available.
Saving/Loading	Allow the user to save the current game and load it at another time.
Hideable GUI	Allow the user to turn the GUI on/off.
Multiple Players	Provide multiple support for multiple players. (Multiple viewports, same keyboard)

Multiple Types of Interactions	Provide multiple types of obstacles that react to a collision with the user's vehicle differently. (E.g. rubber, gravity, etc)
Moving Obstacles	Provide moving obstacles that either animate or follow a discrete path.
Power-Ups	Insert special items within random boxes that modify the state of the user's vehicle or the state of the application.
Enemies	Include enemy objects within the 2D world that attempt to prevent the user from collecting all boxes.

Table 5.0 – Extra Features